Course Goals

• To gain understanding of the basic structure of programming languages:
  – Data types, control structures, naming conventions,...

• To study different language paradigms:
  – Functional (Scheme), imperative (C), logic (Prolog), object-oriented (C++), parallel (OpenMP)
  – To ensure an appropriate language is chosen for a task

• To know the principles underlying all programming languages:
  – To make learning new programming languages easier
  – To enable full use of a programming language
  – To understand the implementation challenges of different programming constructs / features

Programming languages are tools ⇒ understand how to design or use them
Course Information

Prerequisites (summary):

- CS 205 (Introduction to Discrete Structures)
- CS 211 (Computer Architecture)

Important facts:

staff: Prof. Zheng Zhang
      Hongyu Li, Yikai Zhang, TBA

lectures: Wed 9:50am - 11:10am /Fri 2:50pm - 4:10pm
          AB-2160

recitations:
          section 1, Wed 12:15pm-1:10pm, LSH-B267 LIV
          section 2, Fri 10:35am-11:30am, TIL-253
          section 3, Fri 12:15-1:10pm, LSH-B115

Basis for grades (subject to changes):

  10%  homework
  25%  mid-term exam
  35%  final exam (cumulative)
  30%  three major programming projects
  10%  extra-credit homework and project questions
Course Information (Cont.)

- Additional (recommended) texts: see course web page

Course material is available on our class website at

www.cs.rutgers.edu/courses/314/classes/fall_2016_zhang

In addition, there will be a piazza course page: piazza.com/rutgers/fall2016/cs314. All questions and discussion regarding homeworks and projects should be posted here. **PLEASE DO NOT POST DUPLICATE QUESTIONS.**

All programming will be done on the ilab cluster. Get yourself an ilab account (see link on bottom of our 314 website). Learn to do the normal things — edit, compile, ...
Course Information (Cont.)

**Academic Integrity** (see our web page)
→ read-protect your directories and files (ilab)
→ no group projects
→ will use software tools to check code plagiarism

14 weeks, no "make-up" work after the end of the course. If there is a problem, let me know immediately.

**IMPORTANT INFORMATION ⇒** will be posted on 314 web page and/or on piazza page!

- Failure to take a scheduled exam
- Grading of homeworks and projects
- Instructions of how to submit programming projects
- Partial credit for late project submissions

Email TAs or me:

- **Subject line** has to start with 314:, e.g., 314: *Question about my midterm exam*
- **No** project and homework questions; post them on the piazza discussion forums;
Course Information (Cont.)

Special permission numbers

• No SPN will be given due to limited TA resources.
I use detailed lecture notes

• I try to moderate my speed
• You are encouraged to ask questions
• all lecture notes are on the Web (PDF)
• you should still take some notes

I’ll tell you where we are in the book

• I don’t lecture directly from the book
• You need to read the book
• I strongly recommend coming to the lectures
What is the Purpose of a Programming Language?

A programming language is . . .

a set of conventions for communicating an algorithm. *Ellis Horowitz*

Purposes:

- specifying algorithm and data structures
- communicating algorithms among people
- allowing reasoning and establishing correctness
Why Use Anything Besides Machine Code?

This is a C program that uses two one-dimensional arrays \texttt{a} and \texttt{b} of size \texttt{SIZE}. The arrays are initialized, and then a sum reduction is performed. The size of the arrays and the result of the sum reduction is printed out.

\begin{verbatim}
#include <stdio.h>

#define SIZE 100
int main() {
  int a[SIZE], b[SIZE];
  int i, sum;

  for (i=0; i<SIZE; i++) {
    a[i] = 1;
    b[i] = 2;
  }
  sum = 0;
  for (i=0; i<SIZE; i++)
    sum = sum + a[i] + b[i];
  printf("for two arrays of size %d, sum = %d\n", SIZE, sum);
}
\end{verbatim}
Why Use Anything Besides Machine Code?

Compiler: gcc -O3 -S example.c ⇒ example.s

.file "example.c"
.version "01.01"
gcc2_compiled:.
.section .rodata.str1.32,"aMS",@progbits,1
.align 32
.LC0:
.string "for two arrays of size %d, sum = %d\n"
.text
.align 4
.globl main
.type main,@function
main:
pushl %ebp
movl %esp, %ebp
xorl %eax, %eax
subl $808, %esp
movl $99, %edx
.p2align 2
.L21:
movl $1, -408(%ebp,%eax)
movl $2, -808(%ebp,%eax)
addl $4, %eax
decl %edx
jns .L21
xorl %ecx, %ecx
xorl %eax, %eax
movl $99, %edx
.p2align 2
.L26:
addl -408(%ebp,%eax), %ecx
addl -808(%ebp,%eax), %ecx
addl $4, %eax
decl %edx
jns .L26
pushl %eax
pushl %ecx
pushl $100
pushl $.LC0
call printf
addl $16, %esp
leave
ret
.Lfe1:
.size main,.Lfe1-main
.ident "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)"
Why Use Anything Besides Machine Code?

gcc -o example.o -O3 example.c; strip example.o;
objdump -d example.o

objdump: example.o: No symbols

objdump: example.o: file format elf32-sparc

Disassembly of section .text:

00000000 <.text>: 
00000010 bc 10 20 00 clr %fp
00000014 e0 03 a0 40 ld [ %sp + 0x40 ], %l0
00000018 a2 03 a0 44 add %sp, 0x44, %l1
0000001c 9c 23 a0 20 sub %sp, 0x20, %sp
00000020 80 90 00 01 tst %g1
00000024 02 80 00 04 be 0x10468
00000028 90 10 00 01 mov %g1, %o0
0000002c 40 00 40 c4 call 0x20770
00000030 01 00 00 00 nop
00000034 11 00 00 41 sethi %hi(0x10400), %o0
00000038 90 12 22 48 or %o0, 0x248, %o0 ! 0x106d8
0000003c 40 00 40 c0 call 0x20770
00000040 01 00 00 00 nop
00000044 11 00 00 41 sethi %hi(0x10400), %o0
00000048 90 12 22 48 or %o0, 0x248, %o0 ! 0x106d8
0000004c 40 00 40 c0 call 0x20770
00000050 01 00 00 00 nop
00000054 11 00 00 41 sethi %hi(0x10400), %o0
00000058 90 12 22 48 or %o0, 0x248, %o0 ! 0x106d8
0000005c 40 00 40 c0 call 0x20770
00000060 01 00 00 00 nop
00000064 11 00 00 41 sethi %hi(0x10400), %o0
1054c: 13 00 00 00 sethi %hi(0), %o1
10550: 92 12 60 08 or %o1, %o1 ! %o8
10554: 40 00 40 90 call 0x20794
10558: a0 05 c0 09 ld [ %l7 + %o1 ], %o0
1055c: 11 00 00 00 sethi %hi(0), %o0
10560: 92 12 20 0c or %o0, 0xc, %o0 ! 0xc
10564: d0 05 c0 09 ld [ %l7 + %o0 ], %o2
10568: 92 10 20 01 mov 1, %o1
1056c: d2 22 80 00 st %o1, [ %o2 ]
10570: 81 c7 e0 08 ret
10574: 81 e8 00 00 restore
10578: 9d e3 bf 90 save %sp, -112, %sp
1057c: 81 c7 e0 08 ret
10580: 81 e8 00 00 restore
10584: 9d e3 bf 90 save %sp, -112, %sp
10588: 2f 00 00 40 sethi %hi(0x10000), %l7
10590: 7f ff ff cb call 0x104b8
10594: ae 05 e1 90 add %l7, 0x190, %l7 ! 0x10190
10598: 9d e3 bf 90 save %sp, -112, %sp
1059c: 81 c7 e0 08 ret
105a0: 80 a2 60 00 cmp %o1, 0
105a4: 02 80 00 08 be 0x105c4
105a8: 11 00 00 00 sethi %hi(0), %o1
105ac: 2f 00 00 40 sethi %hi(0x10000), %l7
105b0: 9d e3 bf 90 save %sp, -112, %sp
105b4: 81 c7 e0 08 ret
105b8: 9d e3 bf 90 save %sp, -112, %sp
105bc: 81 c7 e0 08 ret
105c0: 80 a2 60 00 cmp %o1, 0
105c4: 02 80 00 08 be 0x105c4
105c8: 9d e3 bf 90 save %sp, -112, %sp
105cc: 81 c7 e0 08 ret
105d0: 9d e3 bf 90 save %sp, -112, %sp
105d4: 9d e3 bf 90 save %sp, -112, %sp
105d8: 9d e3 bf 90 save %sp, -112, %sp
105dc: 9d e3 bf 90 save %sp, -112, %sp
105e0: 9d e3 bf 90 save %sp, -112, %sp
105e4: 9d e3 bf 90 save %sp, -112, %sp
105e8: 9d e3 bf 90 save %sp, -112, %sp
105ec: 9d e3 bf 90 save %sp, -112, %sp
105f0: 9d e3 bf 90 save %sp, -112, %sp
105f4: 9d e3 bf 90 save %sp, -112, %sp
105f8: 9d e3 bf 90 save %sp, -112, %sp
105fc: 9d e3 bf 90 save %sp, -112, %sp
10600: 9d e3 bf 90 save %sp, -112, %sp
10604: 9d e3 bf 90 save %sp, -112, %sp
10608: 9d e3 bf 90 save %sp, -112, %sp
1060c: 9d e3 bf 90 save %sp, -112, %sp
10610: 9d e3 bf 90 save %sp, -112, %sp
10614: 9d e3 bf 90 save %sp, -112, %sp
10618: 9d e3 bf 90 save %sp, -112, %sp
1061c: 9d e3 bf 90 save %sp, -112, %sp
10620: 9d e3 bf 90 save %sp, -112, %sp
10624: 9d e3 bf 90 save %sp, -112, %sp
10628: 9d e3 bf 90 save %sp, -112, %sp
1062c: 9d e3 bf 90 save %sp, -112, %sp
10630: 9d e3 bf 90 save %sp, -112, %sp
10634: 9d e3 bf 90 save %sp, -112, %sp
10638: 9d e3 bf 90 save %sp, -112, %sp
10640: 9d e3 bf 90 save %sp, -112, %sp
10644: 9d e3 bf 90 save %sp, -112, %sp
10648: 9d e3 bf 90 save %sp, -112, %sp
1064c: 9d e3 bf 90 save %sp, -112, %sp
10650: 9d e3 bf 90 save %sp, -112, %sp
10654: 9d e3 bf 90 save %sp, -112, %sp
10658: 9d e3 bf 90 save %sp, -112, %sp
1065c: 9d e3 bf 90 save %sp, -112, %sp
10660: 9d e3 bf 90 save %sp, -112, %sp
10664: 9d e3 bf 90 save %sp, -112, %sp
10668: 9d e3 bf 90 save %sp, -112, %sp
1066c: 9d e3 bf 90 save %sp, -112, %sp
Disassembly of section .init:

000106bc <.init>:
106bc: 9d e3 bf a0 save %sp, -96, %sp
106c0: 7f ff ff b1 call 0x10584
106c4: 01 00 00 00 nop
106c8: 7f ff ff e6 call 0x10660
106cc: 01 00 00 00 nop
106d0: 81 c7 e0 08 ret
106d4: 81 e8 00 00 restore

Disassembly of section .fini:

000106d8 <.fini>:
106d8: 9d e3 bf a0 save %sp, -96, %sp
106dc: 7f ff ff 79 call 0x104c0
106e0: 01 00 00 00 nop
106e4: 81 c7 e0 08 ret
106e8: 81 e8 00 00 restore

Disassembly of section .plt:

00020740 <.plt>:
...
20770: 03 00 00 30 sethi $hi(0xc000), %g1
20774: 30 bf ff f3 b,a 0x20740
20778: 01 00 00 00 nop
2077c: 03 00 00 3c sethi $hi(0xf000), %g1
20780: 30 bf ff f0 b,a 0x20740
20784: 01 00 00 00 nop
20788: 03 00 00 48 sethi $hi(0x12000), %g1
2078c: 30 bf ff e4 b,a 0x20740
20790: 01 00 00 00 nop
20794: 03 00 00 54 sethi $hi(0x15000), %g1
20798: 30 bf ff ea b,a 0x20740
2079c: 01 00 00 00 nop
207a0: 03 00 00 60 sethi $hi(0x18000), %g1
207a4: 30 bf ff e7 b,a 0x20740
207a8: 01 00 00 00 nop
207ac: 03 00 00 6c sethi $hi(0x1b000), %g1
207b0: 30 bf ff e4 b,a 0x20740
207b4: 01 00 00 00 nop
207b8: 01 00 00 00 nop
Why Use Anything Besides Machine Code?

Need for high-level programming languages for

- Readable, familiar notations
- Machine independence (portability)
- Consistency checks during implementation
- Dealing with scale

The art of programming is the art of organizing complexity. Example: *Dijkstra, 1972*

However:

- Acceptable loss of efficiency

  First FORTRAN compiler built by IBM, in 1957, translated into code as efficient as hand-coded code. *John Backus*
Why Learn More than One Programming Language?

- Each language encourages thinking about a problem in a particular way.
- Each language provides (slightly) different expressiveness & efficiency.

⇒ The language should match the problem.

Why Learn About Programming Language PRINCIPLES?

A programming language is a tool.

Studying the design of a tool leads to:

- Better understanding of its functionality and limitations.
- Increased competence in using it.
- Basis for lots of other work in computer science.
Computational Paradigms

Imperative:
Sequence of state-changing actions.

- Manipulate an abstract machine with:
  1. Variables naming memory locations
  2. Arithmetic and logical operations
  3. Reference, evaluate, assign operations
  4. Explicit control flow statements
- Fits the von Neumann architecture closely
- Key operations: Assignment and “Goto”

Functional:
Composition of operations on data.

- No named memory locations
- Value binding through parameter passing
- Key operations: Function application and Function abstraction

Basis in lambda calculus
Computational Paradigms (Cont.)

Logic:

Formal logic specification of problem.

- Programs say *what* properties the solution must have, not *how* to find it
- Solutions through reasoning process.
- Key operation: *Unification*

Basis in *first order predicate logic*

Object-Oriented:

Communication between abstract objects.

- “Objects” collect both the data and the operations
- “Objects” provide *data abstraction*
- Can be either imperative or functional
- Key operation: *Message passing or Method invocation*
Computational Paradigms (Cont.)

Event-Driven:
Objects are associated with events

- events are asynchronous
- arrival of an event triggers action
- main applications: GUI, simulations
- Key operation: event handling

Parallel:
Computations and data accesses at the same time

- task and data parallelism
- different granularities: instruction, loop, or task level
- synchronization: locks, message passing, ...
- Key notions: control and data dependencies
Compilers

source code → compiler → machine code

errors

Implications:

• recognize legal (and illegal) programs
• generate correct code
• manage storage of all variables and code
• need format for object (or assembly) code

Big step up from assembler – higher level notations
Things to Do

Things to do for next lecture:

- read Scott: Chapter 1 (covers today’s lecture)
- read Scott: Chapters 2.1 and 2.2; ALSU: Chapters 3.1 - 3.4
- get an ilab account
- learn to use piazza discussion group

Recitations will start next Week.